Apache Cordova Development Guide.《Apache Cordova 开发指南》

waylau

Published with GitBook



目錄

《Apache Cordova 开发指南》



Apache Cordova Development Guide helps people to learn Cordova step by step with a large number of samples.

Apache Cordova is an open-source mobile development framework. It allows you to use standard web technologies such as HTML5, CSS3, and JavaScript for cross-platform development, avoiding each mobile platforms' native development language. Applications execute within wrappers targeted to each platform, and rely on standards-compliant API bindings to access each device's sensors, data, and network status. The latest version of Cordova is v5.4.1.

There is also a GitBook version of the book: http://waylau.gitbooks.io/cordova-dev-guide. Let's READ!

《Apache Cordova 开发指南》，中文，通过大量实例，图文并茂帮助 Cordova 初学者快速掌握 Cordova。

Apache Cordova 是跨平台应用开发解决方案，而 PhoneGap 就是 Cordova 的原始版本和最流行的发布。包括：

- 使用 HTML, CSS 和 JS 开发移动端应用
- 一次编码支持多平台编译，其中包括 Amazon Fire OS、Android、BlackBerry、Firefox OS、iOS、Ubuntu、Windows Phone、Windows 等
- 免费且开源

Cordova 的最新版本是 5.4.1。

本书业余时间所著，水平有限、时间紧张，难免疏漏，欢迎指正，点此提问。感谢您的参与！

书中所有实例，在https://github.com/waylau/cordova-dev-guide 的 `samples` 目录下。 从目录开始阅读吧！另外有 GitBook 的版本方便阅读 http://waylau.gitbooks.io/cordova-dev-guide。

Contact 联系作者:

- www.waylau.com

快速开始

本文介绍了如何用 Cordova 快速构建一个应用。

安装 Cordova

Cordova 命令行运行于 Node.js 环境，可以通过 NPM，进行安装。

```
$ npm install -g cordova
```

注：国内环境下，可以设置 NPM 镜像，来使安装加速。参见《加速 npm》

创建项目

使用命令行创建一个空项目。切换到任意目录下，执行 `cordova create <path>` 来创建项目，本例的项目名称为 `cordova-getstarted`

```
$ cordova create cordova-getstarted
```

添加平台

项目创建后，切换到项目下，执行 `cordova platform add <platform name>` 来添加你所期望构建项目的平台。本例的平台是 `browser` ,即在浏览器中运行。

执行 `run cordova platform` 可以看到所能执行添加的平台。

```
$ cd cordova-getstarted

$ cordova platform add browser
```

运行 app

执行 `cordova run <platform name>` 即可。

```
$ cordova run browser
```

```
管理员: C:\Windows\system32\cmd.exe

D:\workspaceGithub\cordova-dev-guide\samples>cordova create cordova-getstarted
Creating a new cordova project.

D:\workspaceGithub\cordova-dev-guide\samples>cd cordova-getstarted

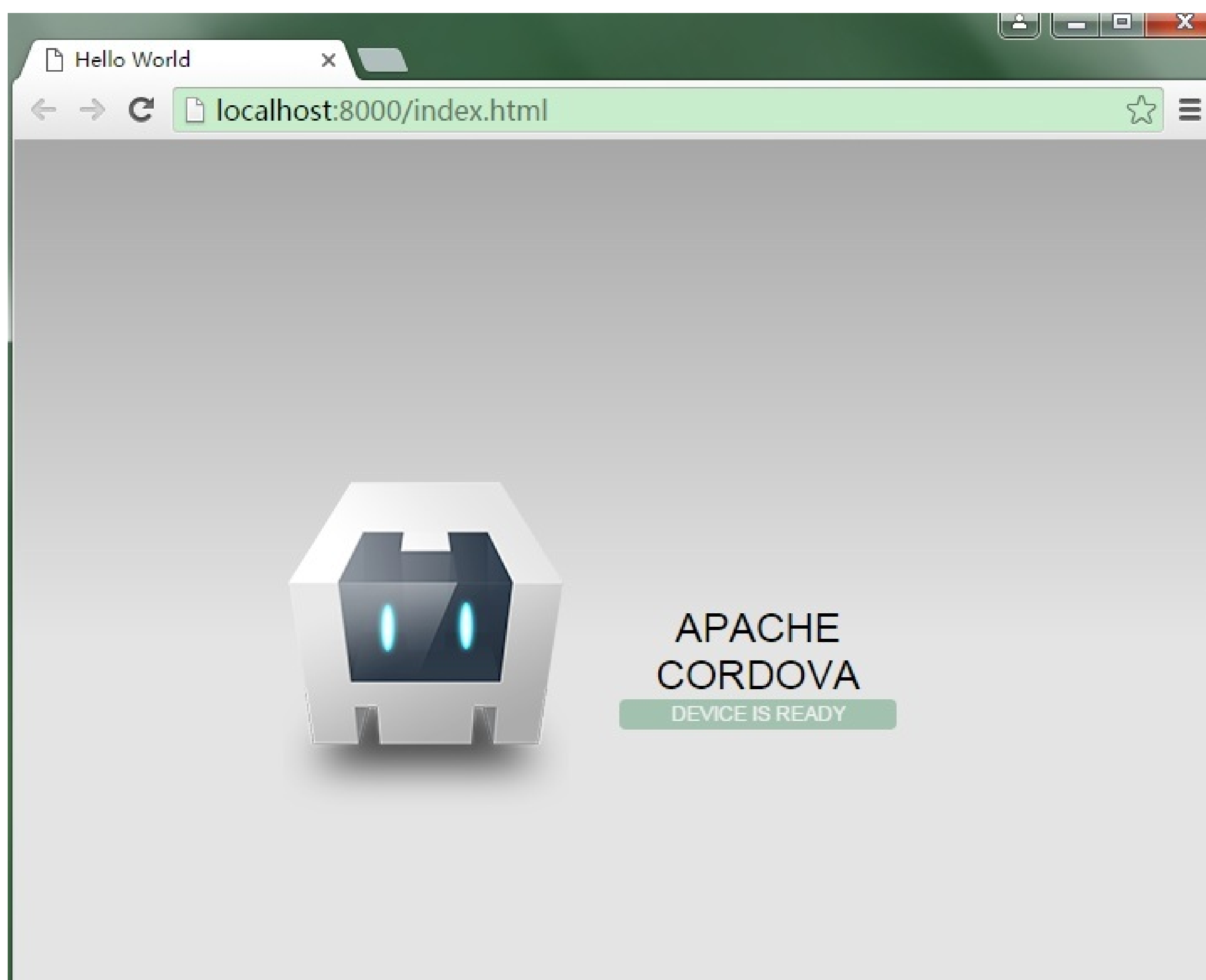
D:\workspaceGithub\cordova-dev-guide\samples\cordova-getstarted>cordova platform
 add browser
Adding browser project...
Running command: cmd "/s /c "C:\Users\Administrator\.cordova\lib\npm_cache\cordo
va-browser\4.0.0\package\bin\create.bat D:\workspaceGithub\cordova-dev-guide\sam
ples\cordova-getstarted\platforms\browser io.cordova.hellocordova HelloCordova""

Creating Browser project. Path: platforms\browser
Discovered plugin "cordova-plugin-whitelist" in config.xml. Installing to the pr
oject
Fetching plugin "cordova-plugin-whitelist@1" via npm
Installing "cordova-plugin-whitelist" for browser

This plugin is only applicable for versions of cordova-android greater than 4.0.
 If you have a previous platform version, you do *not* need this plugin since th
e whitelist will be built in.

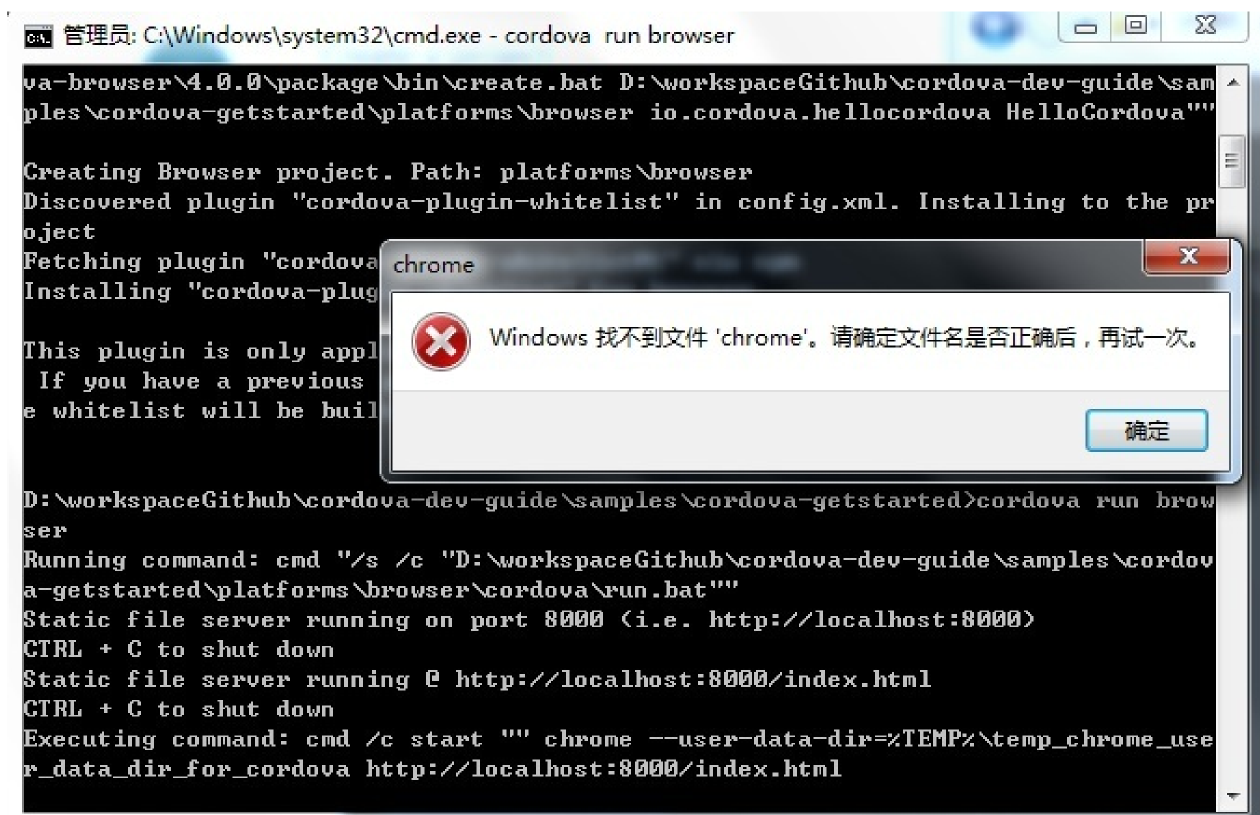

D:\workspaceGithub\cordova-dev-guide\samples\cordova-getstarted>cordova run brow
ser
Running command: cmd "/s /c "D:\workspaceGithub\cordova-dev-guide\samples\cordov
a-getstarted\platforms\browser\cordova\run.bat""
Static file server running on port 8000 (i.e. http://localhost:8000)
CTRL + C to shut down
Static file server running @ http://localhost:8000/index.html
```

浏览器会自动运行，效果如下



问题

问题1：报“找不到chrome”错误



解决方法1：安装 chrome 浏览器即可。

解决方法2：非 chrome 浏览器网址访问 http://localhost:8000/index.html 即可

源码

- 见 https://github.com/waylau/cordova-dev-guide 的 `samples` 目录下 `cordova-getstarted`

总览

Apache Cordova 是开源的移动开发框架。允许你使用标准的 web 技术，如 HTML5、 CSS3 和 JavaScript 来构建跨平台的开发，从而避免使用原生的开发语言。应用针对每个平台进行包装内执行，并依靠符合标准的 API 绑定来访问不同设备的传感器，数据和网络状态。

Apache Cordova 在2012年10月成为 Apache Software Foundation (ASF) 的顶级项目，在 Apache License v2.0 协议下继续保持开源和免费。

Apache Cordova 面向的用户主要有：

- 移动开发者，想要扩展应用程序到多个平台上，而不必用每个平台的语言和工具集重新实现它。
- Web开发人员，希望部署 Web 应用程序打包分发到各种应用程序商店门户。
- 移动开发者，对原生应用程序与 可以访问设备级的 API 的 WebView（特殊的浏览器窗口）组件混合开发感兴趣，或者想开发原生与 WebView 组件之间的插件接口。

基本组件

Cordova 应用程序依赖于一个共同的 config.xml 文件，提供有关应用程序的信息，并指定影响它的工作原理，如，是否响应方向的变化参数。该文件遵循 W3C 打包的 Web应用程序，或插件，规范。

该应用程序本身是一个网页，在默认情况下名为 index.html 的本地文件，即无论引用CSS，JavaScript，图片，多媒体文件或其他资源都需要它来运行。该应用程序执行的本地应用程序包装内的 WebView。

Cordova 使得 WebView 可提供其整个用户界面应用程序。在某些平台上，它也可以是混合了原生应用组件和 WebView 的 hybrid 应用。（见 嵌入 WebViews了解详细信息。）

一个插件接口用于 Cordova 和原生组件相互通信。这使您可以让 JavaScript 调用原生代码。理想情况下，JavaScript API 面向了原生代码在多个设备平台是一致的。3.0 版本，提供了插件绑定标准的设备 API。第三方插件提供额外的绑定功能并不一定适用于所有平台的。您可以在插件注册表中找到这些第三方插件，并可以在你的应用程序使用他们。您也可以开发自己的插件，参考插件开发指南中的介绍。插件是必要的，例如，Cordova 和自定义的原生组件之间的通信。

注意： 3.0 版本，当创建一个 Cordova 项目时，并默认不会有任何插件存在，如果想使用插件，需要显示添加相应的插件。

Cordova 并不提供任何 UI 组件或者 MV* 框架。如果需要，需要自己添加第三方包。

开发路径

3.0 版本，主要提供2种开发移动应用的流程，他们各有优势：

- Cross-platform (CLI) workflow（跨平台（命令行界面）的工作流程）：如果你希望你的应用程序，在尽可能多的不同的移动操作系统上运行，很少需要特定平台开发，那么你适合使用此工作流程。该工作流程围绕 cordova 工具为中心，就是 3.0 中引入的 Cordova CLI。CLI 是一个高层次的工具，可以让您一次搭建多种平台项目，抽象出许多低级别的 shell 脚本的功能。该CLI 复制一套通用的网络资源集注入到子目录每个移动平台中，使得任何必要的配置更改，就运行构建脚本生成应用程序二进制文件。 CLI 还提供了通用接口应用到你的应用程序中。有关CLI的更多详细信息，请参阅命令行界面。除非你有使用以平台为中心的工作流程的必要，否则建议使用跨平台的工作流。
- Platform-centered workflow （平台为中心的工作流程）：如果你想专注于建立一个应用程序的单一平台，并需要能够在较低的水平，对其进行修改，请使用此工作流程。您需要使用这种方法，例如，如果你希望你的应用程序混合使用基于 Web 的Cordova 组件定制的本地组件。作为一个经验法则，如果你需要修改 SDK 中的项目，那么请使用此工作流程。该工作流依赖于一组低级别的 shell 脚本针对每个支持的平台，和一个单独的 Plugman 工具。虽然你可以使用这个流程来构建跨平台的应用程序，但是比较困难，因为缺乏一个更高级别的工具，来适应独立的构建周期和不同平台的插件更新。不过，这个工作流程让您可通过每个 SDK 提供的开发方案更多的机会，而且是复杂的混合应用程序所必不可少的。见 平台开发指南 了解更多详情。

当第一次开始，最容易使用的跨平台的工作流程来创建一个应用程序。然后如果你需要更好地控制 SDK， 您可以切换到以平台为中心的工作流程。低级别的 shell 工具在与 CLI 类似的单独的发布包中。对于由 CLI 最初生成的项目，这些 shell 工具也都在项目的 `platforms/*/cordova` 目录中找到。

注意：一旦你从 CLI 的工作流程来转到特定于平台的 SDK 和 shell 工具为中心，你不能转回去了。该 CLI 维护一套通用的跨平台的源代码，基于它来改写特定平台的源代码。为了保护您对特定平台的资源的任何修改，你需要切换到该平台为中心的 shell 工具，依赖于特定平台的源代码而忽略了跨平台的源代码。

安装 Cordova

不同的流程有不同的安装方式：

- Cross-platform workflow: 见 命令行界面
- Platform-centered workflow: 见 平台开发指南

安装之后，建议你针对你需要开发的平台，对照 平台开发指南。同时建议你查看隐私指南，安全指南，以及后续步骤。对于配置 Cordova，请参阅 config.xml 文件。对于从 JavaScript 一个访问设备上的本地函数，请参考插件API。

命令行界面（CLI）

本文介绍如何使用 命令行界面（CLI）创建应用程序，并将它们部署到各种原生移动平台。这个工具允许你创建新的项目，在不同的平台构建，并运行在实际设备或仿真器中。 CLI 是用于在概述中描述的跨平台的工作流的主要工具。当然，你也可以使用 CLI 来初始化项目代码，然后切换到不同的平台的SDK和 shell 工具作为后续发展。

前提

使用 CLI 前，需安装目标平台的 SDK 。（详见平台开发指南）

CLI 支持平台情况：

- iOS (Mac)
- Amazon Fire OS (Mac, Linux, Windows)
- Android (Mac, Linux, Windows)
- BlackBerry 10 (Mac, Linux, Windows)
- Windows Phone 8 (Windows)
- Windows (Windows)
- Firefox OS (Mac, Linux, Windows)

注：对于只支持 Windows 的平台，可以在 Mac 机子上运行 Windows 的虚拟机，或者双启动形式运行 Windows。详见 Windows Phone 8 平台指南 和 Windows 平台指南

安装 Cordova CLI

安装 Cordova CLI 步骤如下：

1. 下载安装 Node.js
2. 下载安装 git 客户端。由于被墙，国内用户从官网下载困难，可异步至 Git for Windows 国内下载站
3. 使用 `npm` 安装 `cordova` 模块。国内用户可以通过设置镜像来加速安装，详见加速 npm

对于 OS X 和 Linux:

```
$ sudo npm install -g cordova
```

对于 Windows:

```
C:\>npm install -g cordova
```

创建应用

切换到源代码的目录，然后运行命令，如下所示：

```
$ cordova create hello com.waylau.cordova.hello HelloWorld
```

这可能需要一些时间命令完成，请耐心等待。命令行 -d 选项会显示进度信息。

第一个参数 `hello` 指定你的项目要生成的一个目录。这个目录原来不存在，Cordova 会为您创建它。其子目录 `www` 容纳您的应用程序的主页，以及在 `css` ，`js` 和 `img` 的各种资源，遵循共同的 Web 开发文件命名约定。这些资源将被存储在设备上的本地文件系统，而不是在远程服务上。该 `config.xml` 文件中包含有需要生成和分发应用程序的重要元数据。

第二个参数输入 `com.waylau.cordova.hello` 提供了一个项目的反向域名风格的标识。此参数是可选的，如果省略该参数，则必须省略第三个参数。您可以之后在 `config.xml` 文件编辑这个值，但建议您一开始就选择合适的值，因为 `config.xml` 生产代码会用到这个值，就像 Java 的包名一样。若不设置此值，则默认值是 `io.cordova.hellocordova` 。

第三个参数的 HelloWorld 提供了应用程序的显示标题。此参数是可选的。您可以之后在 `config.xml` 文件编辑这个值，但建议您一开始就选择合适的值，因为 `config.xml` 生产代码会用到这个值，就像 Java 的包名一样。若不设置此值，则默认值是 `io.cordova.hellocordova` 。默认值是 `HelloCordova` 。

添加平台

所有后续命令都需要在项目的目录下，或者在其范围内的任何子目录中运行：

```
$ cd hello
```

在您可以构建项目前，你需要指定一组目标平台。您运行这些命令的能力取决于你的机器是否支持每个 SDK，以及是否已经安装每个 SDK。在 Mac 运行：

```
$ cordova platform add ios
$ cordova platform add amazon-fireos
$ cordova platform add android
$ cordova platform add blackberry10
$ cordova platform add firefoxos
```

Windows 运行，其中 wp 是指不同版本的 Windows Phone 操作系统的：

```
$ cordova platform add wp8
$ cordova platform add windows
$ cordova platform add amazon-fireos
$ cordova platform add android
$ cordova platform add blackberry10
$ cordova platform add firefoxos
```

查看当前支持的平台：

```
$ cordova platforms ls
```

（注意 `platform` 和 `platforms` 的命令是同义的。）

运行下列命令的代名词，移除一个平台：

```
$ cordova platform remove blackberry10
$ cordova platform rm amazon-fireos
$ cordova platform rm android
```

（注意 `remove` 和 `rm` 的命令是同义的。）

运行命令添加或删除平台，会影响了项目的 `platforms` 目录，该目录下为每个指定的平台显示其子目录的内容。 `www` 的源目录转载到每个平台的子目录中，例如 `platforms/ios/www` 或 `platforms/android/assets/www` 。由于 CLI 会不断地从源 `www` 复制文件，你应该只编辑这些源文件，而不是那些 `platforms` 下的子目录。如果您使用的版本控制软件，你应该把这个源 `www` 文件夹，随着 `merges` 文件夹一起添加到你的版本控制系统。 （关于 `merges` 文件夹的更多信息，可以在下面“使用 merges 自定义每个平台”中找到）

警告：当使用 CLI 来构建应用程序，你不应该在 `/platforms/` 目录编辑任何文件，除非你知道自己在做什么，或者说明文件另有说明。当应用程序构建或者插件准备重新安装时，该目录中的文件是经常会被覆盖的。

如果你意识到了这点，你可以使用一个 SDK，如用 Eclipse 或 Xcode 来打开你创建的项目。您将需要从 `/platforms/` 目录下打开资源来开发SDK。这是因为SDK特定的元数据文件存储在相应的 `/platforms/` 子目录中。 （请参阅平台开发指南了解如何在每个IDE中开发应用程序。）如果你只是想用 CLI 初始化一个项目，然后切换到 SDK 的原生工作，那么请使用此方法。

如果你想在整个开发周期使用跨平台的工作流方法（CLI），请继续往下阅读。

构建应用程序

默认的， `cordova create` 脚本将会创建一个基于web 应用程序的骨架，主页是 `www/index.html` 文件。

迭代的构建程序，运行

```
$ cordova build
```

可以制定构建的平台

```
$ cordova build ios
```

`cordova build` 是下面命令的简化：

```
$ cordova prepare ios
$ cordova compile ios
```

在这种情况下，一旦运行了 `prepare` ，你可以用苹果的 Xcode 的 SDK，以替代修改和编译 Cordova 生成的 `platforms/ios` 特定于平台的代码。您可以使用同样的做法在其他平台的 SDK 中。

在模拟器或者设备上测试应用

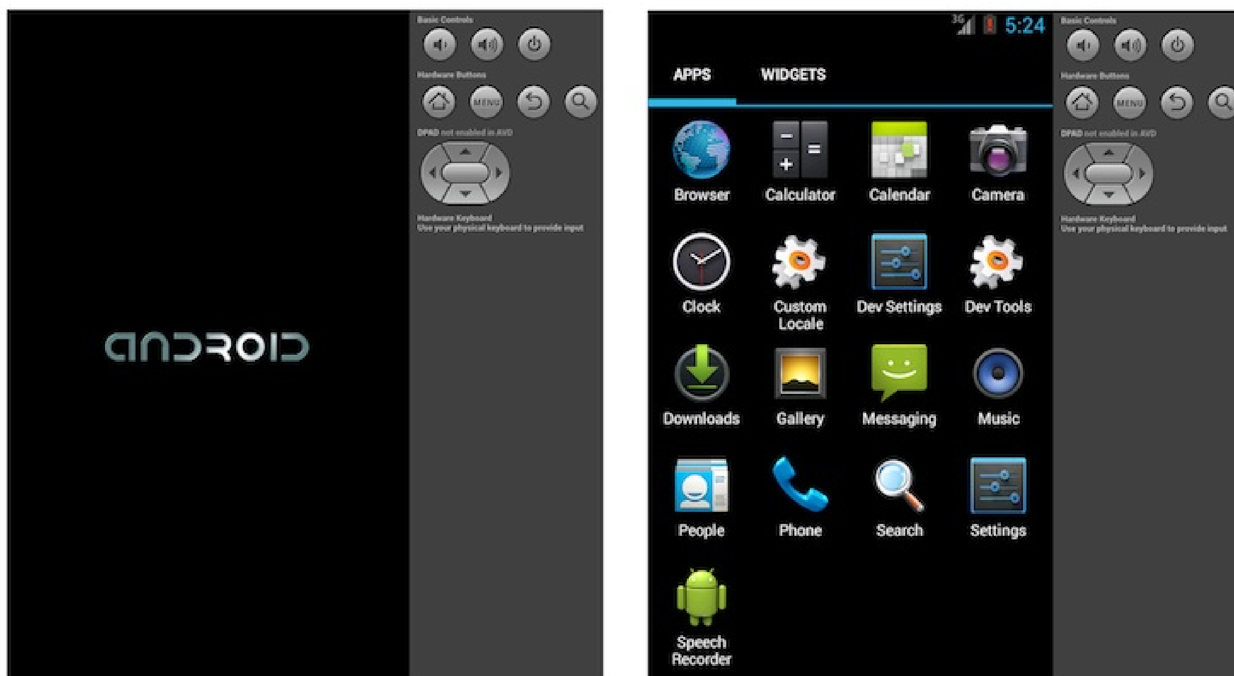
SDK 为移动平台往往捆绑模拟器来执行设备的图像，这样就可以从主屏幕启动应用程序，查看它是如何与众多平台功能进行交互的。运行如下命令，在特定平台构建应用，查看在模拟器中执行效果：

```
$ cordova emulate android
```

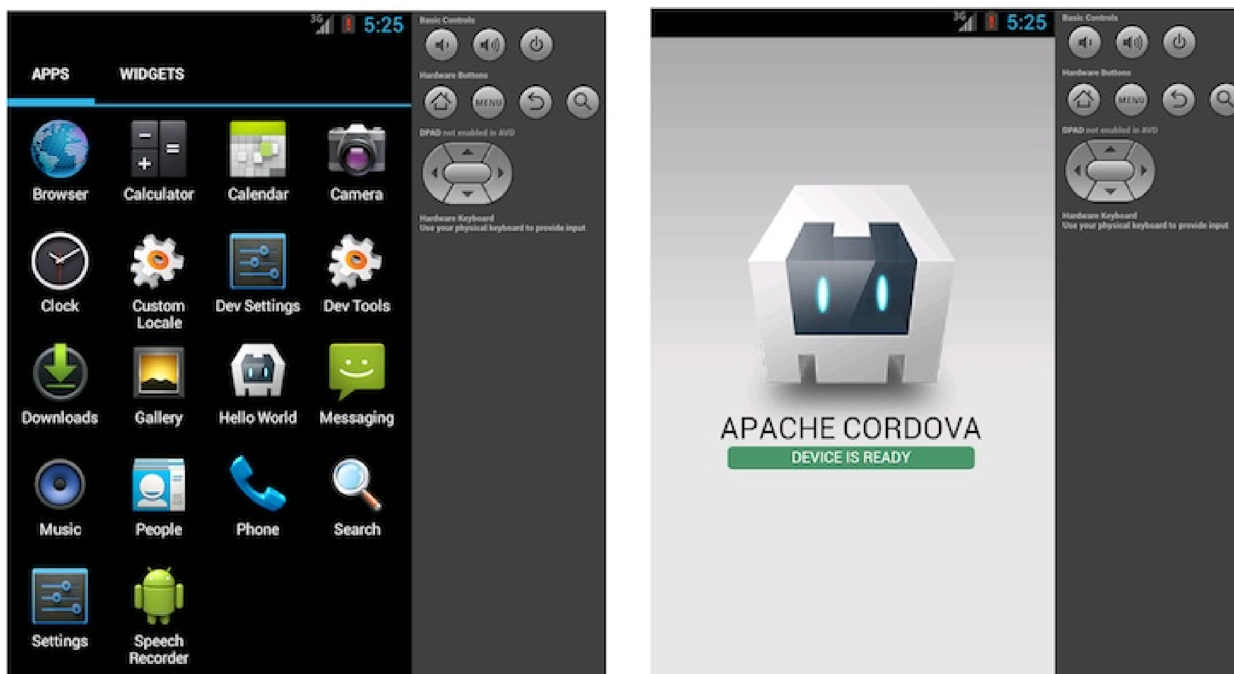
一些移动平台上在默认情况下会模拟一个特定的设备，比如 iOS 项目默认是 iPhone。对于其他平台，您可能需要首先将设备与模拟器相关联。

注：模拟器的支持目前不适用于 Amazon Fire OS。

（请参阅平台开发指南了解详细信息。）例如，您可以先运行 `android` 命令来启动Android SDK，然后运行特定设备的图像，而根据它的默认行为启动它：



接着用 `cordova emulate` 命令刷新模拟器图像显示最新的应用程序，现在是可以从主屏幕启动：



或者，你可以将手机插入电脑，并直接测试应用程序：

```
$ cordova run android
```

在执行上述命令前，先要在设备上做设置。在 Android 和 Amazon Fire OS 上，需要先启用“USB debugging”选项。 请参阅平台开发指南了解各个平台的设置。

添加插件

插件，可以使应用获得与设备级别交流的能力，提供了原生组件的接口。你可以自己设计插件，如，设计一个混合了 Cordova WebView 和原生组件的 hybrid 应用（详见 * 嵌入 WebViews 和 插件开发指南）

3.0 版本后，创建一个 Cordova 是不会存在任何插件的，你需要按照需要显示的添加插件。

可以在 http://plugins.cordova.io/ 查找插件，其中也包含第三方的插件。CLI 也能支持插件的查找。例如使用关键字 `bar` 、 `code` 进行查找。

```
$ cordova plugin search bar code

com.phonegap.plugins.barcodescanner - Scans Barcodes
```

如果只是使用关键字 `bar` ，则查找结果为

```
cordova-plugin-statusbar - Cordova StatusBar Plugin
```

`cordova plugin add` 命令是用来添加插件的，比如

- 基本设备信息(设备 API):

```
$ cordova plugin add cordova-plugin-device
```

- 网络连接和电池事件:

```
$ cordova plugin add cordova-plugin-network-information
$ cordova plugin add cordova-plugin-battery-status
```

- 加速度计，指南针，和地理位置:

```
$ cordova plugin add cordova-plugin-device-motion
$ cordova plugin add cordova-plugin-device-orientation
$ cordova plugin add cordova-plugin-geolocation
```

- 相机，媒体播放和捕捉：

```
$ cordova plugin add cordova-plugin-camera
$ cordova plugin add cordova-plugin-media-capture
$ cordova plugin add cordova-plugin-media
```

- 在设备或者网络上访问文件(File API):

```
$ cordova plugin add cordova-plugin-file
$ cordova plugin add cordova-plugin-file-transfer
```

- 通过对话框或振动通知：

```
$ cordova plugin add cordova-plugin-dialogs
$ cordova plugin add cordova-plugin-vibration
```

- 联系方式:

```
$ cordova plugin add cordova-plugin-contacts
```

- 全球化:

```
$ cordova plugin add cordova-plugin-globalization
```

- 闪屏:

```
$ cordova plugin add cordova-plugin-splashscreen
```

- 打开一个新的浏览器窗口 (InAppBrowser):

```
$ cordova plugin add cordova-plugin-inappbrowser
```

- 控制台调试:

```
$ cordova plugin add cordova-plugin-console
```

注：CLI 增加适合各个平台的插件代码。如果你想开发具有较低级别的 shel l工具或平台的 SDK，你需要运行 Plugman 工具来为每个平台单独添加的插件。（请参阅使用 Plugman 来管理插件。）

`plugin ls` (或 `plugin list`，或 `plugin`）查看当前已安装的插件。以唯一标识做为显示：

```
$ cordova plugin ls    # or 'plugin list'
[ 'cordova-plugin-console' ]
```

移除时，也用唯一标识来移除

```
$ cordova plugin rm cordova-plugin-console
$ cordova plugin remove cordova-plugin-console    # same
```

可以批量添加或者移除：

```
$ cordova plugin add cordova-plugin-console cordova-plugin-device
```

高级插件选项

添加插件时，`@` 制定版本：

```
$ cordova plugin add cordova-plugin-console@latest
$ cordova plugin add cordova-plugin-console@0.2.1
```

非 `registry.cordova.io` 注册的插件，也从其他 git 库添加：

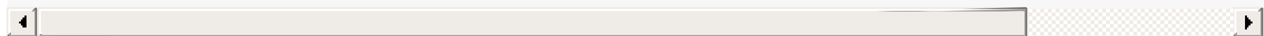
```
$ cordova plugin add https://github.com/apache/cordova-plugin-console.git
```

`#` 来指定一个标签（tag）

```
$ cordova plugin add https://github.com/apache/cordova-plugin-console.git#r0.2.0
```

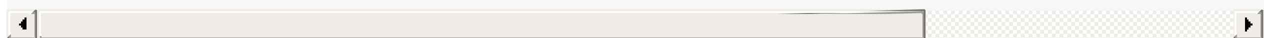
或者一个分支：

```
$ cordova plugin add https://github.com/apache/cordova-plugin-console.git#CB-8438cordova-
```



或者是一个提交：

```
$ cordova plugin add https://github.com/apache/cordova-plugin-console.git#f055daec45575bf
```



如果插件（以及 plugin.xml 文件 ），是在 git 库的子目录，使用 `:`

```
$ cordova plugin add https://github.com/someone/aplugin.git#:/my/sub/dir
```

当然也能做如下合并：

```
$ cordova plugin add https://github.com/someone/aplugin.git#r0.0.1:/my/sub/dir
```

或者，指定一个包含 `plugin.xml` 文件的本地插件目录：

```
$ cordova plugin add ../my_plugin_dir
```

使用 merges 自定义每个平台

虽然 Cordova 让您可以轻松地部署应用程序在许多不同的平台上，有时你需要添加一些自定义功能。在这种情况下，你不希望修改各种 `platforms` 目录下的 `www` 目录中的源文件，因为他们经常会被顶级 `www` 目录的跨平台源所取代。

取而代之的是，顶层 `merges` 目录提供指定资源来部署在特定平台上的地方。 `merges` 镜像中的每个特定平台的子目录反映了 `www` 源代码树的目录结构，使您能够覆盖或根据需要添加文件。例如，下面演示了如何使用 `merges` 以提高 Android 和Amazon Fire OS 设备的默认字体大小：

- 编辑 `www/index.html` , 添加额外的 CSS 文件 `overrides.css` :

```
<link rel="stylesheet" type="text/css" href="css/overrides.css" />
```

- 可选创建一个空的 `www/css/overrides.css` 文件, 应用于非 Android 的构建, 防止文件丢失的错误。
- 在 `merges/android` 下创建一个 `css` 子目录，再加入相应的 `overrides.css` 文件。在 `www/css/index.css` 中指定的字体大小，例如：

```
body { font-size:14px; }
```

在重构项目中，Android 版本采用了自定义字体大小，而其他的维持不变。

还可以使用的 `merges` 来添加不存在于原始 `www` 目录中的文件。例如，一个应用程序可以将一个“后退按钮”图形插入iOS界面，存储在 `merges/ios/img/back_button.png` ， 而 Android 版本，可以改为从相应的硬件按钮事件捕捉 `[backbutton](../../cordova/events/events.backbutton.html)` 。

帮助命令

遇到问题，请呼唤“帮助”。执行：

```
$ cordova help
$ cordova        # same
```

此外，你可以得到一个更详细的帮助。 例如

```
$ cordova run --help
```

`info` 命令产生的潜在有用的信息，如当前已安装的平台和插件，每个平台 SDK 版本，CLI 和 Node.js的版本信息列表：

```
$ cordova info
```

它既在屏幕上展示信息，并输出到本地info.txt。

注：目前，仅适用于 iOS 和 Android 平台的详细信息。

更新 Cordova 和你的项目

使用如下命令更新 `cordova`：

```
$ sudo npm update -g cordova
```

安装特定版本

```
$ sudo npm install -g cordova@3.1.0-0.2.0
```

运行 `cordova -v` 来查看当前运行的版本， to see which version is currently running. 运行 `npm info` 来获取当前版本以及可用版本的信息:

```
$ npm info cordova
```

Cordova 3.0 是支持本节所述的命令行界面的第一个版本。如果您是从之前的版本更新到 3.0，则需要如上所述创建一个新的项目，那么旧的应用程序的资源复制到顶层 `www` 目录。有关升级到 3.0 的进一步的细节，参考平台开发指南。一旦您升级到 `cordova` 的命令行界面，并使用 `npm update` 保持同步。

Cordova 3.0+ 可能仍然会有各种变化，包括项目级目录结构和其他依赖。在运行 `npm` 命令来更新 Cordova 本身后，你可能需要确保项目的资源是符合最新版本的要求。运行命令，如下：

```
$ cordova platform update android
$ cordova platform update ios
...etc.
```

源码

- 本文所用源码，见 https://github.com/waylau/cordova-dev-guide 的 `samples` 目录下 `hello`

平台开发指南

Cordova 支持如下平台：

- Amazon Fire OS
- Android
- BlackBerry 10
- Firefox OS
- iOS
- Ubuntu
- Windows Phone 8
- Windows
- Tizen

开发工具和每个移动平台设备 API 的最新的平台支持情况，可参见 http://cordova.apache.org/docs/en/latest/guide/support/index.html。这里列出的设备 API 是核心插件提供的，其他 API 可通过第三方插件提供。

平台开发

首先安装 CLI，参见命令行界面（CLI）。

其次，针对不同的平台，需要安装相应的 SDK。

下面列出的每个平台的指南会告诉你如何设置每个平台的开发环境：从哪里获得 SDK，如何设置设备模拟器，如何连接设备直接测试，以及如何管理签名密钥的要求。其他指南提供了有关每个平台的独特的配置选项，说明如何添加插件，如何升级每个平台以及特定平台的命令行工具，作为一个低级别的 `cordova` 命令行工具的替代。

Android 平台

本文将介绍了如何设置 SDK 环境来部署 Cordova 应用到 Android 设备上，以及在你的开发流程中如何选择使用 Android 命令行工具。不管你是以平台为中心的工作流程还是跨平台（命令行界面）的工作流程都需要安装 Android SDK。

需求及支持

Cordova 开发 Android 需要 Android SDK，可以安装在 OS X, Linux 或 Windows 操作系统。详见 Android SDK 系统需求.

Cordova 支持 Android 4.0.x（Android API 14 开始） 及更高版本。原则是，Cordova 不再支持 市场占有率少于 5% 的 Android 版本（可参见 Goolge 看板）。Android 版本早于 Android API 10 及 3.x 版本占有率少于 5%。

安装 Java Development Kit (JDK)

安装 JDK 7 或更高版本。

Windows 系统，需设置 `JAVA_HOME` 值为 JDK 的安装路径。如， `C:\Program Files\Java\jdk1.7.0_75` ）

安装 Android SDK

安装 Android Stand-alone SDK Tools 或 Android Studio。 若需要开发新的 Cordova 的 Android 插件，或者使用原生工具来运行或者调试 Android 平台，那么选择 Android Studio。否则 Android Stand-alone SDK Tools 足够用了。

需要将 SDK 的 `tools` 和 `platform-tools` 目录放到你的 `PATH` 中。

- Mac 或 Linux ：修改 `~/.bash_profile` 文件，添加 SDK安装信息，如下： `export PATH=${PATH}:/Development/android-sdk/platform-tools:/Development/android-sdk/tools`
- Windows：修改环境变量 `PATH` ，添加 SDK安装信息，如下： `;C:\Development\android-sdk\platform-tools;C:\Development\android-sdk\tools`

安装 SDK 包

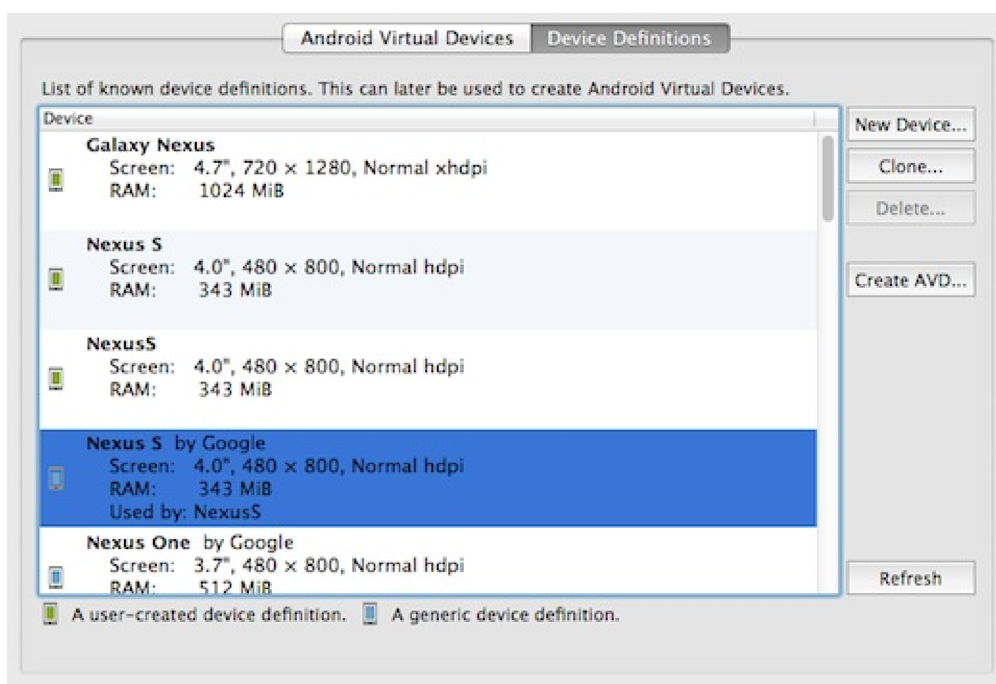
打开 Android SDK Manager (例如，通过终端: `android`) 安装如下内容：

- Android 5.1.1 (API 22) platform SDK
- Android SDK Build-tools version 19.1.0 or higher
- Android Support Repository (Extras)

细节可以参考http://developer.android.com/sdk/installing/adding-packages.html

配置模拟器

Android sdk 没有提供默认的模拟器，需要自己配置。命令行运行 `android` ,点击 `Tools → Manage AVDs` 选择 `Device Definitions`



点击 Create AVD ，选择修改名称，点击 OK

此时，AVD 将出现在 Android Virtual Devices 列表中：

打开模拟器，选择 AVD 并点击 Start。

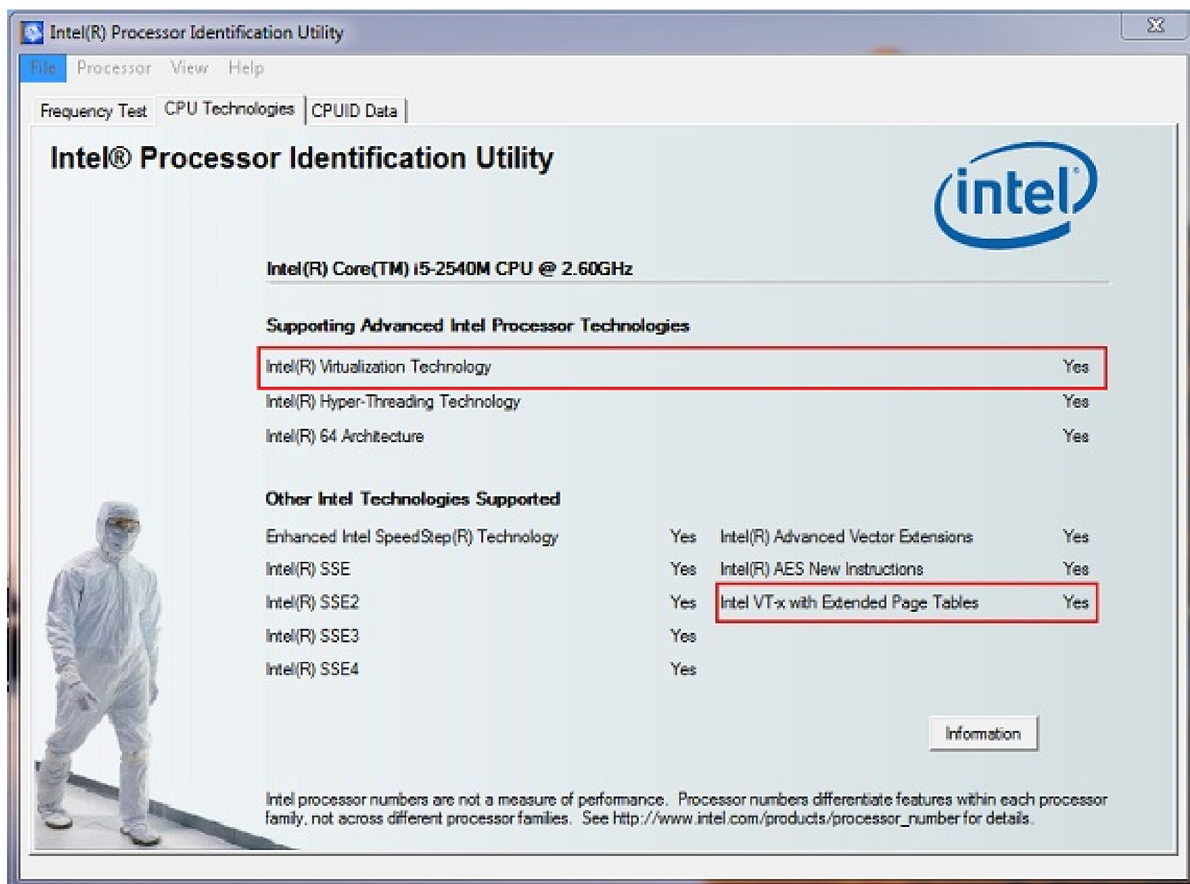


使用 `Virtual Machine Acceleration` 可以加速虚拟机，获得更好的体验。先确认你的系统是否支持如下该技术：

- Intel Virtualization Technology (VT-x, vmx) → Intel VT-x 支持处理器的列表
- AMD Virtualization (AMD-V, SVM), 支持 Linux (自 2006 年5月, 所有 CPU AMD 包含 AMD-V, 除了 Sempron)

Windows 环境下，另外一个查找 Intel 处理器支持 VT-x 技术，是通过执行 `Intel Processor Identification Utility` ，你可以在 https://downloadcenter.intel.com/Detail_Desc.aspx?ProductID=1881&DwnldID=7838 下载该工具。或者使用 `OS Independent` ,在 https://downloadcenter.intel.com/Detail_Desc.aspx?ProductID=1881&DwnldID=7840&lang=eng 下载该工具。

执行 `Intel Processor Identification Utility` ，可看到如下界面：



为加速模拟器，下载至少一个 `Intel x86 Atom System Image`，如 `Intel Hardware Accelerated Execution Manager (HAXM)`

打开 Android SDK Manager，选择如下：

Name	API	Rev.	Status
☐ Android 4.4 (API 19)			
☐ Documentation for Android SDK	19	2	Not installed
☐ Samples for SDK	19	3	Not installed
☑ Intel x86 Atom System Image	19	1	Not installed
☐ Google APIs	19	2	Not installed
☐ Sources for Android SDK	19	2	Not installed
☐ Extras			
☐ Android Support Repository		4	Not installed
☐ Google Analytics App Tracking SDK		3	Not installed
☐ Google Play services for Froyo		12	Not installed
☐ Google Play services		14	Not installed
☐ Google Repository		5	Not installed
☐ Google Play APK Expansion Library		3	Not installed
☐ Google Play Billing Library		5	Not installed
☐ Google Play Licensing Library		2	Not installed
☐ Google Web Driver		2	Not installed
☑ Intel x86 Emulator Accelerator (HAXM)		3	Not installed

下载后，可以执行 Android SDK 中的 `extras/intel/Hardware_Accelerated_Execution_Manager` 的 Intel 安装包，安装过程，遇到问题，可以查阅 http://software.intel.com/en-us/android/articles/speeding-up-the-android-emulator-on-intel-architecture

步骤如下：

1. 安装至少一个 `Intel x86 Atom System Image` ,如 `Intel Hardware Accelerated Execution Manager`
2. 运行 `Intel` 安装包，在 Android SDK 的 `extras/intel/Hardware_Accelerated_Execution_Manager`
3. 创建新的 AVD 目标设为这个 Intel image
4. 当启动模拟器, 确保加载 HAX 模块时没有错误

创建项目

创建项目可以是选择 CLI 或者是 Android 特有的 shell 工具。使用 CLI 如下：

```
$ cordova create hello com.waylau.cordova.hello HelloWorld
$ cd hello
$ cordova platform add android
$ ccordova prepare              # or "cordova build"
```

Unix 、Windows 环境下，使用低级别的 shell 工具:

```
$ /path/to/cordova-android/bin/create /path/to/new/hello com.waylau.cordova.hello HelloWo
C:\path\to\cordova-android\bin\create.bat C:\path\to\new\hello com.waylau.cordova.hello H
```

构建项目

若使用 CLI，则项目的顶级 `www` 目录包含的就是源文件。构建执行如下：

```
$ cordova build                   # build all platforms that were added
$ cordova build android           # build debug for only Android
$ cordova build android --debug   # build debug for only Android
$ cordova build android --release # build release for only Android
```

若使用 Android 特有的 shell 工具，则有些不同。生成项目，默认的应用源码是在 `assets/www` 子目录。执行 `build` 将会清空项目文件，从新构建。以下是 Mac 和 Windows 下的语法，分别用于 debug 和 release

```
$ /path/to/project/cordova/build --debug
C:\path\to\project\cordova\build.bat --debug

$ /path/to/project/cordova/build --release
C:\path\to\project\cordova\build.bat --release
```

部署应用

使用 CLI，部署应用到模拟器或者设备

```
$ cordova emulate android       #to deploy the app on a default android emulator
$ cordova run android --device  #to deploy the app on a connected device
```

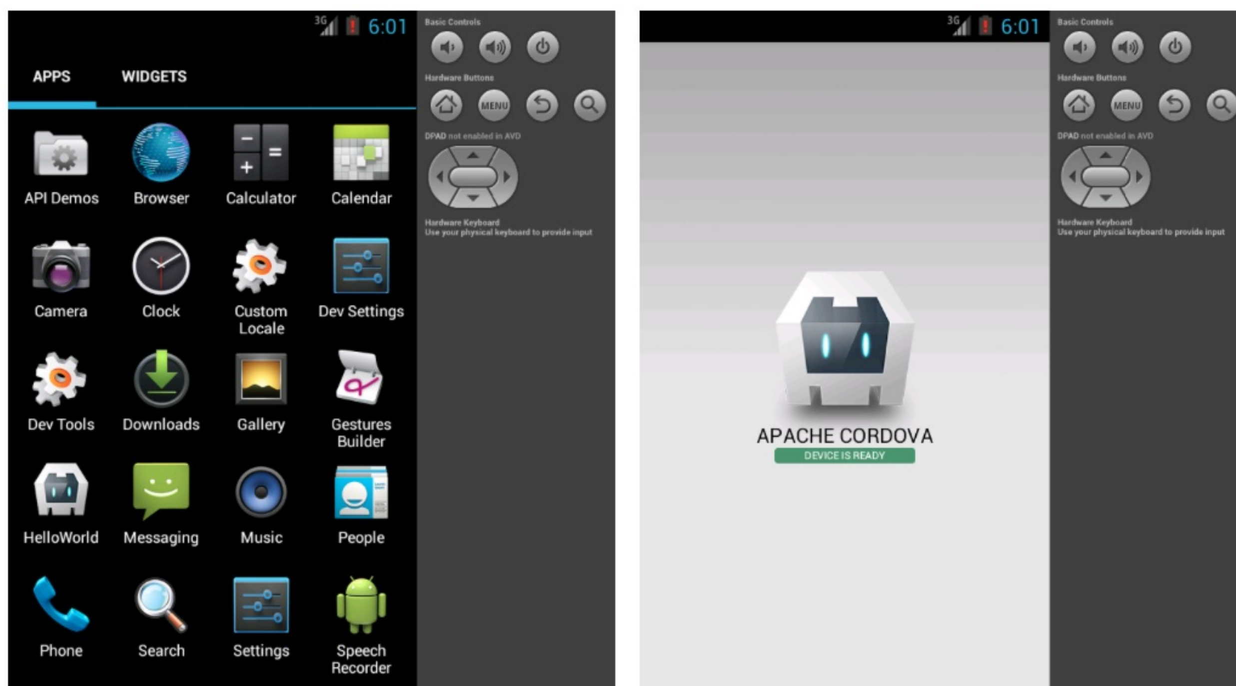
否则，使用备用 shell 界面：

```
$ /path/to/project/cordova/run --emulator
$ /path/to/project/cordova/run --device
```

可以使用 `cordova run android --list` 查看所有存在的目标， `cordova run android --target=target_name` 来运行特定的模拟器或者是设备(如 `cordova run android --target="Nexus4_emulator"`)

使用 `cordova run --help` 来查看其他的构建和运行选项。

启动应用，界面如下：



执行 `run` 它默认就 `build` 了项目，其他附加参数有 `--debug`，`--release`，和 `--nobuild`，来控制是否需要重新构建

```
$ /path/to/project/cordova/run --emulator --nobuild
```

其他命令

生成日志：

```
$ /path/to/project/cordova/log
C:\path\to\project\cordova\log.bat
```

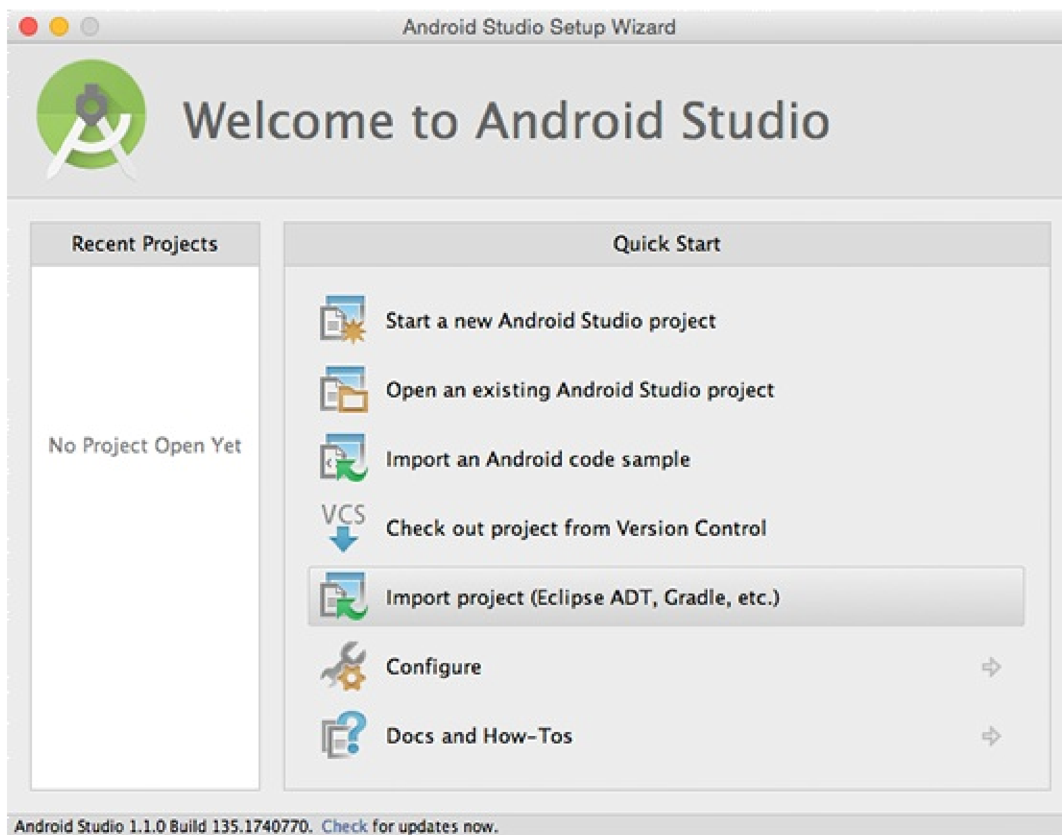
清除项目:

```
$ /path/to/project/cordova/clean
C:\path\to\project\cordova\clean.bat
```

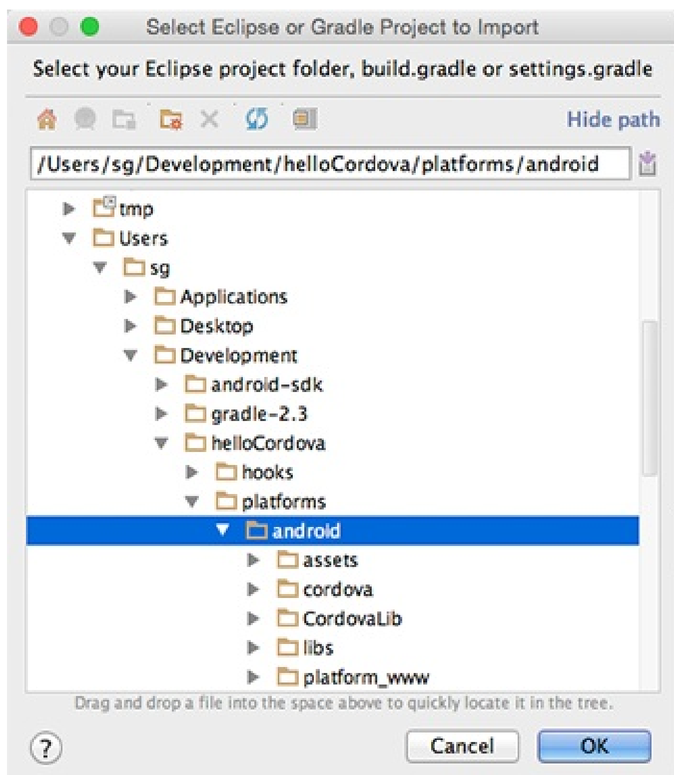
在 SDK 中打开新项目

当项目中已经添加了 android 平台，则可以用 Android Studio 打开：

1. 启动 Android Studio
2. 选择 Import Project (Eclipse ADT, Gradle, etc)

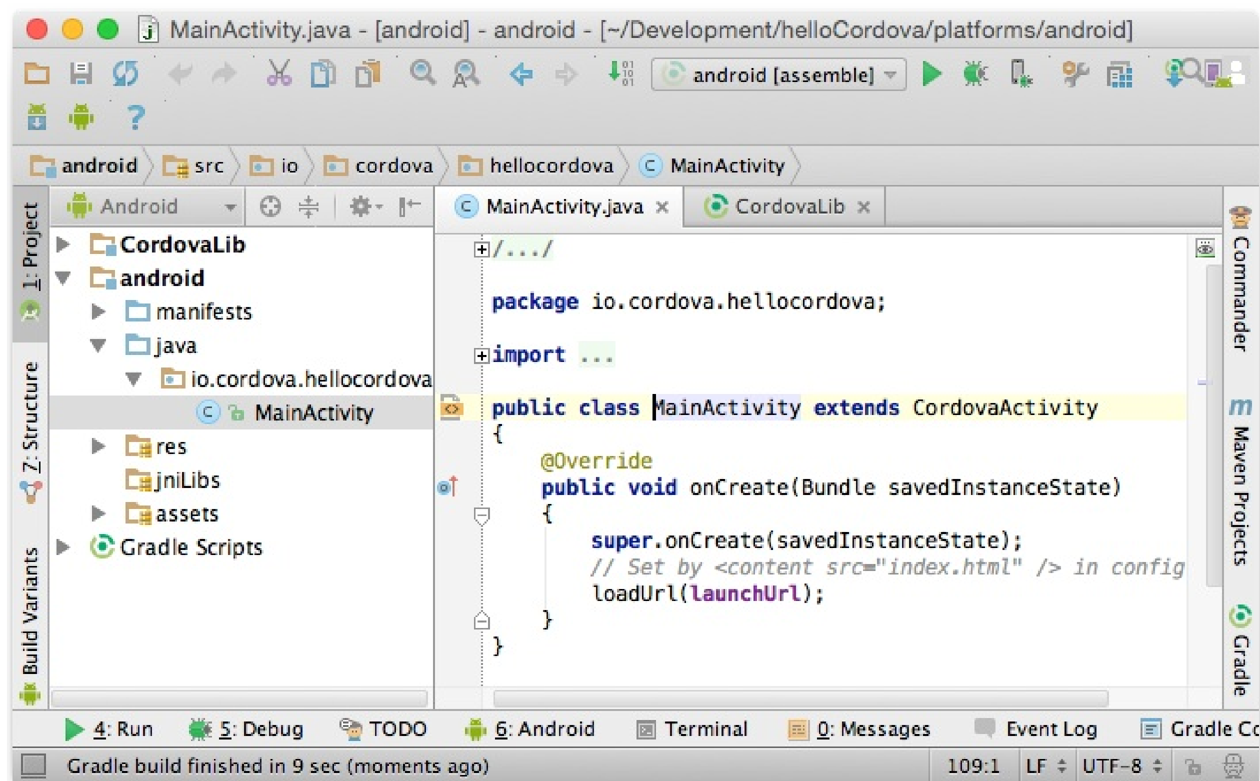


1. 选择 android 平台存储的位置 (`your/project/platforms/android`)



1. `Gradle Sync` 的问题，直接回答 Yes

此时，可以从 Android Studio 直接构建和运行应用了



更多细节，可以参考 Android Studio Overview 和 Building and Running from Android Studio

Android shell 工具

本文展示了如何使用以平台为中心的 shell 工具来开发 Android 应用。这些工具允许你创建、构建和运行 Android 应用。

创建项目

执行 `create` 命令，在 Mac/Linux 和 Windows 的语法如下：

```
$ /path/to/cordova-android/bin/create /path/to/project com.example.project_name ProjectNa

C:\>\path\to\cordova-android\bin\create.bat \path\to\project com.example.project_name Pro
```

构建

执行这个，先会清空项目而后才构建。

Debug, 在 Mac/Linux 和 Windows 的语法如下：

```
$ /path/to/project/cordova/build --debug

C:\>\path\to\project\cordova\build.bat --debug
```

Release, 在 Mac/Linux 和 Windows 的语法如下：

```
$ /path/to/project/cordova/build --release

C:\>\path\to\project\cordova\build.bat --release
```

运行

`run` 命令可以接受如下可选参数：

- 指定 Target，包括： `--emulator` , `--device` , 或 `--target=<targetID>`
- 指定 Build，包括： `--debug` , `--release` , 或 `--nobuild`

```
$ /path/to/project/cordova/run [Target] [Build]

C:\>\path\to\project\cordova\run.bat [Target] [Build]
```

请确保你至少创建一个 Android Virtual Device,，否则系统会提示您用 `android` 命令这样做。如果有多个 AVD 可作为一个目标，系统会提示您选择一个。默认情况下运行命令检测已连接的设备，或当前正在运行的模拟器（如果没有设备被发现）。

对应用签名

Android 应用签名的需求，可查看 http://developer.android.com/tools/publishing/app-signing.html

签名一个应用，需要如下参数：

- Keystore (--keystore): 存储 key 的编译文件的路径
- Keystore password (--storePassword): keystore 密码
- Alias (--alias): 指定用于签名的私钥 id
- Password (--password): 指定密钥的密码
- Type of the keystore (--keystoreType): pkcs12, jks (默认值：自动检测基于文件扩展名)

这些参数可以在 `build` 或 `run` 脚本里面指定。

另外，您也可以使用（`--buildConfig`）参数在构建配置文件（build.json）中指定。下面是一个构建配置文件的示例：

```
{
     "android": {
         "debug": {
             "keystore": "..\android.keystore",
             "storePassword": "android",
             "alias": "mykey1",
             "password" : "password",
             "keystoreType": ""
         },
         "release": {
             "keystore": "..\android.keystore",
             "storePassword": "",
             "alias": "mykey2",
             "password" : "password",
             "keystoreType": ""
         }
     }
 }
```

对于发行的签名，密码可以被排除在外，构建系统会发出提示，要求输入密码。

也有支持混合和匹配的命令行参数和 build.json 文件的参数。命令行参数的值将获得优先权。这对于在命令行上指定密码是有用的。

日志

```
$ /path/to/project/cordova/log

C:\>\path\to\project\cordova\log.bat
```

清除

```
$ /path/to/project/cordova/clean

C:\>\path\to\project\cordova\clean.bat
```

使用 Gradle 构建

自 cordova-android@4.0.0，项目的构建使用 Gradle。对于 Gradle 的学习，可以参阅《Gradle 2 用户指南》 。

Gradle 属性

下面属性可以在构建时自定义：

- cdvBuildMultipleApks (默认: false)

如果这样设置，那么多个 APK 文件，将生成：每一个库项目（X86，ARM等）支持本地平台生成一个。这可能是重要的，如果你的项目使用了大量本地库，它可以显着提高所产生的 APK 的大小。

如果没有设置，则一个单个的 APK 将生成可以在所有设备中使用。

- cdvVersionCode

覆盖 AndroidManifest.xml 中的 versionCode。

- cdvReleaseSigningPropertiesFile (默认: release-signing.properties)

包含签名信息的发布构建的 `.properties` 文件路径，类似于:

```
storeFile=relative/path/to/keystore.p12
storePassword=SECRET1
storeType=pkcs12
keyAlias=DebugSigningKey
keyPassword=SECRET2
```

`storePassword` 和 `keyPassword` 是可选的，如果省略会提示。

- cdvDebugSigningPropertiesFile (默认: debug-signing.properties)

和 cdvReleaseSigningPropertiesFile 一样,但是为了调试构建。在需要和其他开发者分享签名密钥时非常有用

- cdvMinSdkVersion

覆盖 AndroidManifest.xml 中的 minSdkVersion。当创建基于 SDK 版本多个 APK 时非常有用

- cdvBuildToolsVersion

重载自动检测 android.buildToolsVersion 值

- cdvCompileSdkVersion

重载自动检测 android.compileSdkVersion 值

扩展 build.gradle

自定义 build.gradle, 不是直接编辑，而是应该创建一个同级文件名为 `build-extras.gradle` 。该文件将被包含在主 `build.gradle` 。下面是一个例子：:

```
# Example build-extras.gradle
# This file is included at the beginning of `build.gradle`
ext.cdvDebugSigningPropertiesFile = '../../android-debug-keys.properties'
# When set, this function allows code to run at the end of `build.gradle`
ext.postBuildExtras = {
    android.buildTypes.debug.applicationIdSuffix = '.debug'
}
```

注意插件也可以包含进来，通过在 `build-extras.gradle` 文件设置:

```
<framework src="some.gradle" custom="true" type="gradleReference" />
```

构建示例

```
export ORG_GRADLE_PROJECT_cdvMinSdkVersion=14
cordova build android -- --gradleArg=-PcdvBuildMultipleApks=true
```

Android 配置

`config.xml` 文件控制应用在各个应用和 CordovaWebView 实例的基本设置。本文只涉及 Android 构建部分。详见 config.xml 文件。

- `KeepRunning` (boolean, 默认 `true`): 确定项目是否在触发 `[pause](../../../cordova/events/events.pause.html)` 事件后在后台运行。设置为 `false` 并不会在 `[pause](../../../cordova/events/events.pause.html)` 事件后杀掉应用,应用程序是在后台而只是暂停 cordova webview 的执行。

  ```
  <preference name="KeepRunning" value="false"/>
  ```

- `LoadUrlTimeoutValue` (数值是毫秒, 默认值是 `20000` , 20 秒): 当加载页面时，超时会抛出错误。下面是一个设置为10秒的例子

  ```
  <preference name="LoadUrlTimeoutValue" value="10000"/>
  ```

- `SplashScreen` (string, 默认是 `splash`): 该文件减去其在 `res/drawable` 目录扩展的名称。各种资源必须在不同的子目录共享这一名字。

  ```
  <preference name="SplashScreen" value="mySplash"/>
  ```

- `SplashScreenDelay` (数值是毫秒, 默认值是 `3000`): 闪屏图像显示的时间。

  ```
  <preference name="SplashScreenDelay" value="10000"/>
  ```

- `InAppBrowserStorageEnabled` (boolean, 默认是 `true`): 控制InAppBrowser 默认的浏览器打开的页面是否可以访问相同的 localStorage 和 WebSQL 存储。

  ```
  <preference name="InAppBrowserStorageEnabled" value="true"/>
  ```

- `LoadingDialog` (string, 默认是 `null`): 如果设置，加载应用程序的第一页时，对话框显示指定的标题和消息，以及一个旋转器。标题和消息之间用逗号分离出来，并在显示对话框之前这个逗号被删除。

  ```
  <preference name="LoadingDialog" value="My Title,My Message"/>
  ```

- `LoadingPageDialog` (string, 默认是 `null`): 与 `LoadingDialog` 类似, 但在第一个页面之后加载所有页面

```
<preference name="LoadingPageDialog" value="My Title,My Message"/>
```

- `ErrorUrl` (URL, 默认是 `null`):如果设置，则会显示一个错误页面来代替"Application Error"对话框.

```
<preference name="ErrorUrl" value="myErrorPage.html"/>
```

- `ShowTitle` (boolean, 默认是 `false`): 显示标题在屏幕上方

```
<preference name="ShowTitle" value="true"/>
```

- `LogLevel` (string, 默认是 `ERROR`): 设置日志级别，值包括 `ERROR` , `WARN` , `INFO` , `DEBUG` , 和 `VERBOSE` .

```
<preference name="LogLevel" value="VERBOSE"/>
```

- `SetFullscreen` (boolean, 默认是 `false`): 与全局配置文件 xnl 中的 `Fullscreen` 参数类似。该属性在未来版本将会移除，不推荐使用

- `AndroidLaunchMode` (string, 默认是 `singleTop`): 设置 Activity `android:launchMode` 属性。这个会在应用图标启动或者已经运行在运行时发生变化，值包括 `standard` , `singleTop` , `singleTask` , `singleInstance` .

```
<preference name="AndroidLaunchMode" value="singleTop"/>
```

- `DefaultVolumeStream` (string, 默认是 `default` , 在 cordova-android 3.7.0 版本加入): 设置硬件音量按钮链接到哪个音量。默认情况下，这是“call”链接到手机，而“media”链接的是平板电脑。设置“media”到你的应用程序的音量按钮将会随时改变媒体音量。需要注意的是使用 Cordova 的媒体插件时，当任何媒体对象是活动时，音量按键会动态变化，以控制媒体音量。

- `OverrideUserAgent` (string, 默认不设置):如果设置，该值将取代 webview 的老用户代理。从应用程序/浏览器请求远程页面时，对于确定该请求是有用的。请谨慎使用这个设置，因为可能会导致与 Web 服务器的兼容性问题。对于大多数情况下，使用 AppendUserAgent 代替。

```
<preference name="OverrideUserAgent" value="Mozilla/5.0 My Browser" />
```

- `AppendUserAgent` (string, 默认不设置):如果设置，该值将追加到的 webview 的老用户代理的尾部。当 OverrideUserAgent 使用，该值将被忽略。

  ```
  <preference name="AppendUserAgent" value="My Browser" />
  ```

Android 插件

本节详细介绍了如何在 Android 平台上的实现原生插件代码。在读这篇文章前，请参阅应用程序插件关于插件的结构和共同的 JavaScript 接口的概述。本节展示“回声”插件示例，来实现 Cordova webview 与原生平台的通信交互。该示例也可见 CordovaPlugin.java.

基于 Cordova-Android 的 Android 插件，是 Android 的 WebView 与连接到其挂钩。插件被表示为在 config.xml 文件类的映射。一个插件至少有一个Java类，是继承了 CordovaPlugin 类，覆盖它的 `execute` 方法之一。作为最佳实践，该插件还应该处理任何消息的插件之间传递的 `[pause](../../../cordova/events/events.pause.html)` 和 `[resume](../../../cordova/events/events.resume.html)` 事件。插件长时间运行的请求，后台活动，如媒体播放，监听器，或内部状态应该实现 `onReset()` 方法为好。它在 `WebView` 导航到新页面或刷新时执行，从而重新加载 JavaScript。

插件 类 映射

插件的 JavaScript 接口使用 `cordova.exec` 方法如下:

```
exec(<successFunction>, <failFunction>, <service>, <action>, [<args>]);
```

请求从 WebView 发起到 Android 端，调用 `service` 类的 `action` 方法，传递 `args` 数组参数。

发布一个插件，无论是 Java 文件或者是 *jar* 文件，插件必须在你的 Cordova-Android 应用的 `res/xml/config.xml` 文件指定。见应用插件如何使用 `plugin.xml` 文件来插入 `feature` 元素:

```
<feature name="<service_name>">
    <param name="android-package" value="<full_name_including_namespace>" />
</feature>
```

service 名称在 JavaScript 的 `exec` 调用中相匹配。该值是 Java 类的完全限定命名空间的标识。否则，该插件即使可以编译，仍无法在 Cordova 使用。

插件的初始化及有效期

插件对象实例由每个 `WebView` 的有效期内创建。插件不会初始化，直到第一个引用被 JavaScript 调用, 除非 `config.xml` 的 `<param>` 中的 `onload` 的 `name` 属性设置为 `"true"`，如:

```
<feature name="Echo">
    <param name="android-package" value="<full_name_including_namespace>" />
    <param name="onload" value="true" />
</feature>
```

插件的 `initialize` 方法启动如下：

```
@Override
public void initialize(CordovaInterface cordova, CordovaWebView webView) {
    super.initialize(cordova, webView);
    // your init code here
}
```

编写 Android Java 插件

插件类的 `execute` 方法，实现方式一般如下:

```
    @Override
    public boolean execute(String action, JSONArray args, CallbackContext callbackContext
        if ("beep".equals(action)) {
            this.beep(args.getLong(0));
            callbackContext.success();
            return true;
        }
        return false;  // Returning false results in a "MethodNotFound" error.
    }
```

JavaScript `exec` 方法的 `action` 参数对应私有类的可选参数的调用。

当捕捉异常并返回错误，JavaScript 的错误应该尽可能与 Java 异常相匹配。

线程

插件的 JavaScript 并不运行在 `WebView` 接口的主线程，而是运行在 `WebCore` 线程的 `execute` 方法。如果需与其他用户界面交互，使用方法修改如下：

```
@Override
public boolean execute(String action, JSONArray args, final CallbackContext callbackC
    if ("beep".equals(action)) {
        final long duration = args.getLong(0);
        cordova.getActivity().runOnUiThread(new Runnable() {
            public void run() {
                ...
                callbackContext.success(); // Thread-safe.
            }
        });
        return true;
    }
    return false;
}
```

当不需在主界面线程运行，又不想阻塞 `WebCore` 线程，使用如下:

```
@Override
public boolean execute(String action, JSONArray args, final CallbackContext callbackC
    if ("beep".equals(action)) {
        final long duration = args.getLong(0);
        cordova.getThreadPool().execute(new Runnable() {
            public void run() {
                ...
                callbackContext.success(); // Thread-safe.
            }
        });
        return true;
    }
    return false;
}
```

添加依赖库

如果插件需额外的库，需要在 `config.xml` 添加，

方法1 通过 *Gradle* 引用，比如:

```
<framework src="com.android.support:support-v4:+" />
```

详情可参见 *Gradle* 的依赖管理，参考《Gradle 2 用户指南》

方法2 *JAR* 文件替换插件的文件夹使用 `lib-file` 来链接,比如:

```
<lib-file src="src/android/libs/gcm.jar"/>
```

用此方法的前提是你非常清楚你特定插件的依赖 jar 不会被其他插件所使用，不然会代码构建的问题。

Echo Android Plugin 示例

本例演示 JavaScript 接口的 *echo* 功能，使用 `plugin.xml` 来注入 `feature` 到 `config.xml` 文件:

```
<platform name="android">
    <config-file target="config.xml" parent="/*">
        <feature name="Echo">
            <param name="android-package" value="org.apache.cordova.plugin.Echo"/>
        </feature>
    </config-file>
</platform>
```

添加 `src/org/apache/cordova/plugin/Echo.java` 文件:

```
package org.apache.cordova.plugin;

import org.apache.cordova.CordovaPlugin;
import org.apache.cordova.CallbackContext;

import org.json.JSONArray;
import org.json.JSONException;
import org.json.JSONObject;

/**
 * This class echoes a string called from JavaScript.
 */
public class Echo extends CordovaPlugin {

    @Override
    public boolean execute(String action, JSONArray args, CallbackContext callbackCon
        if (action.equals("echo")) {
            String message = args.getString(0);
            this.echo(message, callbackContext);
            return true;
        }
        return false;
    }

    private void echo(String message, CallbackContext callbackContext) {
        if (message != null && message.length() > 0) {
            callbackContext.success(message);
        } else {
            callbackContext.error("Expected one non-empty string argument.");
        }
    }
}
```

文件继承 `CordovaPlugin`，并覆盖 `execute()` 方法，来接收 `exec()` 的消息。`execute()` 方法首先测试 `action` 的值，在这种情况下，只有一个有效 `echo` 值。任何其他操作都返回 `false`，并导致一个 `INVALID_ACTION` 错误，给 JavaScript 回调。

接下来，该方法检索使用 `args` 对象的 `getString` 方法，指定传递给该方法的第一个参数。值传递给私有 `echo` 方法后，对参数进行检查，以确保它不是 `null` 或空字符串，在这种情况下 `callbackContext.error()` 调用 JavaScript 的错误回调。如果各种检查都通过，`callbackContext.success()` 把原始 `message` 字符串返回到 JavaScript 的成功回调作为参数。

Android 集成

Android 功能 `Intent` 允许与其他进程进行交互。插件访问 `CordovaInterface` 对象，而该对象可以访问应用的 Android `Activity` 。这个 `Context` 需要启动一个新的 Android `Intent` 。 `CordovaInterface` 允许插件来启动一个 `Activity` 用于处理结果，并当 `Intent` 返回给应用时设置回调插件。

Cordova 2.0 插件不能直接访问 `Context` , 遗留的 `ctx` 成员不再推荐使用。所有的 `ctx` 方法存在于 `Context` , 所以 `getContext()` 和 `getActivity()` 可以返回所需的对象。

Android 权限

Android 权限是在安装时而不是在运行时的进行处理。这些权限需要在使用时在应用程序中声明，并且这些权限需要被添加到 Android Manifest。这可以通过使用在config.xml 注入在 AndroidManifest.xml 文件中来实现这些权限。下面的示例为使用联系人权限。

```
<config-file target="AndroidManifest.xml" parent="/*">
    <uses-permission android:name="android.permission.READ_CONTACTS" />
</config-file>
```

Android 权限 (Cordova-Android 5.0.x 及跟高版本)

Android 6.0 "Marshmallow" 引入了新的权限模型，可以在需要时启动后者关闭权限。他意味着应用程序处理这些权限更改必须是面向未来的，这是Cordova-Android 5.0 版本的重点。

需要在运行时处理权限可以在 Android 开发者文档中找到。

至于一个插件而言，权限请求通过权限方法，该方法的签名如下：

```
cordova.reqquestPermission(CordovaPlugin plugin, int requestCode, String permission);
```

为减少冗长，标准的做法是分配一个本地静态变量：

```
public static final String READ = Manifest.permission.READ_CONTACTS;
```

同时定义一个请求编码，如下:

```
public static final int SEARCH_REQ_CODE = 0;
```

在 exec 方法,权限将会被检查:

```
        if(cordova.hasPermission(READ)) {
            search(executeArgs);
        }
        else
        {
            getReadPermission(SEARCH_REQ_CODE);
        }
```

此时，我们只需调用 requestPermission:

```
protected void getReadPermission(int requestCode)
{
    cordova.requestPermission(this, requestCode, READ);
}
```

这会调用 activity 并会提示要求权限。若用户有权限，则结果必须被 onRequestPermissionResult 方法处理。这是每个插件必须覆盖的。示例如下:

```
public void onRequestPermissionResult(int requestCode, String[] permissions,
                                         int[] grantResults) throws JSONException
{
    for(int r:grantResults)
    {
        if(r == PackageManager.PERMISSION_DENIED)
        {
            this.callbackContext.sendPluginResult(new PluginResult(PluginResult.Status.ER
            return;
        }
    }
    switch(requestCode)
    {
        case SEARCH_REQ_CODE:
            search(executeArgs);
            break;
        case SAVE_REQ_CODE:
            save(executeArgs);
            break;
        case REMOVE_REQ_CODE:
            remove(executeArgs);
            break;
    }
}
```

权限也可以用数组形式，下面是 Geolocation 插件:

```
String [] permissions = { Manifest.permission.ACCESS_COARSE_LOCATION, Manifest.permission
```

当请求权限时，只需如下处理:

```
cordova.requestPermissions(this, 0, permissions);
```

这就要求在数组中指定权限。这是一个好主意，提供可公开访问的权限数组，因为这可以由使用你的插件作为依赖插件，虽然这不是必需的。

调试 Android 插件

Android 的调试可以使用 Eclipse 或 Android Studio 来完成，虽然推荐使用 Android Studio。由于 Cordova-Android 目前用作一个库项目，并且插件支持源代码，这使得在 Cordova 应用中调试 Java 代码就像是原生的 Android 应用。

Android WebViews

本文演示了如何在大型 Android 应用里面，内嵌 Cordova-enabled WebView 组件。欲了解组件之间如何交互，参见插件开发指南。

若你对 Android 还不熟悉，可以先参考Android 平台，并安装 Android SDK。 Cordova 1.9 开始, Android 平台依赖 `CordovaWebView` 组件,该组件是基于 `CordovaActivity` 组件构建的。

1. 首先确保有最新的 Cordova 的发布包，从 cordova.apache.org 下载，并解压 Android 包。
2. 切换到 Android 包的 `/framework` 目录，并运行 `ant jar`，它将创建 Cordova `.jar` 文件，格式如 `/framework/cordova-x.x.x.jar`
3. 拷贝 `.jar` 文件到 Android 项目的 `/libs` 目录
4. 添加如下 `/res/xml/main.xml` 文件,设置相应的 `layout_height`, `layout_width` 和 `id`：

```
<org.apache.cordova.CordovaWebView
    android:id="@+id/tutorialView"
    android:layout_width="match_parent"
    android:layout_height="match_parent" />
```

5. 修改 activity 让其实现 `CordovaInterface`，它会实现包含的方法。可以拷贝自 `/framework/src/org/apache/cordova/CordovaActivity.java`, 或者自己实现。下面的代码片段显示，依赖于接口上的基本应用。请注意引用视图 `id` 与上面显示的 XML 片段指定的id属性如何相匹配:

```
public class CordovaViewTestActivity extends Activity implements CordovaInterface {
    CordovaWebView cwv;
    /* Called when the activity is first created. */
    @Override
    public void onCreate(Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.main);
        cwv = (CordovaWebView) findViewById(R.id.tutorialView);
        Config.init(this);
        cwv.loadUrl(Config.getStartUrl());
    }
```

6. 若应用使用了摄像头，用法如下:

```
@Override
public void setActivityResultCallback(CordovaPlugin plugin) {
    this.activityResultCallback = plugin;
}
/**
 * Launch an activity for which you would like a result when it finished. When this
 * your onActivityResult() method is called.
 *
 * @param command           The command object
 * @param intent            The intent to start
 * @param requestCode       The request code that is passed to callback to identify
 */
public void startActivityForResult(CordovaPlugin command, Intent intent, int request
    this.activityResultCallback = command;
    this.activityResultKeepRunning = this.keepRunning;

    // If multitasking turned on, then disable it for activities that return results
    if (command != null) {
        this.keepRunning = false;
    }

    // Start activity
    super.startActivityForResult(intent, requestCode);
}

@Override
/**
 * Called when an activity you launched exits, giving you the requestCode you starte
 * the resultCode it returned, and any additional data from it.
 *
 * @param requestCode       The request code originally supplied to startActivityFor
 *                          allowing you to identify who this result came from.
 * @param resultCode        The integer result code returned by the child activity t
 * @param data              An Intent, which can return result data to the caller (v
 */
protected void onActivityResult(int requestCode, int resultCode, Intent intent) {
    super.onActivityResult(requestCode, resultCode, intent);
    CordovaPlugin callback = this.activityResultCallback;
    if (callback != null) {
        callback.onActivityResult(requestCode, resultCode, intent);
    }
}
```

7. 最后，添加线程池，除非插件没有线程来运行:

```
@Override
public ExecutorService getThreadPool() {
    return threadPool;
}
```

8. 拷贝应用的 HTML 和 JavaScript 文件到 Android 项目的 `/assets/www` 目录
9. 从 `/framework/res/xml` 拷贝 `config.xml` 文件到 `/res/xml` 目录

升级 Android

本文介绍了如何修改 Android 项目来从 旧版的 Cordova 升级。大部分指令适用于与旧的 `cordova` CLI 工具创建的项目。有关 CLI 内容，可以参阅命令行界面（CLI）

升级至 4.0.0

升级至 4.0.0 有些特殊的步骤，首先， 对于非 CLI 项目，运行:

```
    bin/update path/to/project
```

对于 CLI 项目:

1. 升级 `cordova` CLI 版本，见命令行界面（CLI）
2. 在现有的项目中运行 `cordova platform update android` 。

升级 Whitelist

所有的 whitelist 功能现在都通过插件实现，当升级至 4.0.0 后，将不会受 whitelist 保护。Cordova 有2个 whitelist 插件，来提供不同级别的保护：

1. `cordova-plugin-whitelist` 插件 *(推荐)*
    - 这个是值得推荐的，因为比之前的 whitelist 版本更加安全和可配置。
    - 详见 cordova-plugin-whitelist
    - 运行: `cordova plugin add cordova-plugin-crosswalk-webview`
2. `cordova-plugin-legacy-whitelist` 插件
    - 与之前的 whitelist 版本类似，见 cordova-plugin-legacy-whitelist *不会有配置上的变化，但比推荐的插件少一些保护
    - 运行: `cordova plugin add cordova-plugin-legacy-whitelist`

使用 Crosswalk WebView

默认，应用继续使用设备提供的系统 WebView。若想替代之，使用 Crosswalk 插件:

```
cordova plugin add cordova-plugin-crosswalk-webview
```

升级至 Splashscreen 插件

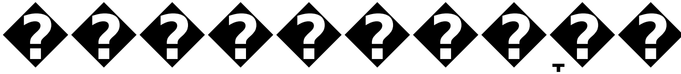
若想使用闪屏，需使用该插件，配置方法没有变，只需如下方法添加插件:

```
cordova plugin add cordova-plugin-splashscreen
```

其他旧版升级

可参阅 http://cordova.apache.org/docs/en/latest/guide/platforms/android/upgrade.html

嵌入 WebViews



使用 Plugman 来管理插件

附录

参考引用

- Apache Cordova Guides
- 《Gradle 2 用户指南》](https://github.com/waylau/Gradle-2-User-Guide)
- Android Developers